

Paper 375-2011

In Control of Your Data Warehouse Processes With the Help of SAS[®] Stored Processes and the SAS[®] Information Delivery Portal

Ralf Vaessen, APG Asset Management, The Netherlands

Danny Pannemans, Bl'n'GO B.V.B.A., Belgium

Juan Quesada, QSD B.V.B.A., Belgium

Koen Vyverman, SAS Institute B.V., The Netherlands

ABSTRACT

In the current economic climate, external and internal pressure on the quality of information management processes is high and still on the rise. In the industry of asset management for pension funds in particular, external regulators require detailed insight into exposure and risk. Keywords are control, transparency, auditability, evidence, data-quality, time-to-market, and security. In this paper, we describe a set of SAS[®] tools for the operational management of data integration and reporting processes at APG (one of the world's leading pension fund management companies): real-time monitoring of data integration jobs; generation and publication of multi-client reports; data quality controls for completeness, consistency and accuracy; and the monitoring of report consumption. These tools are serviced via SAS[®] Stored Processes, executed from within the SAS[®] Information Delivery Portal, and used by members of the APG data warehouse team. We provide a functional overview of the tools and discuss the more interesting aspects of their technical design. We also present an overview of the report catalog available to the various APG business units.

INTRODUCTION

APG Asset Management manages the pension fund capital of approximately 4.5 million Dutch people working in education, government, construction, cleaning and window-cleaning companies, housing corporations, energy and utility companies. A pension fund is any plan, fund, or scheme which provides retirement income. The clients of APG which we refer to in this paper are collective and solidarity-based pension systems that offer Defined Benefit (DB) pension fund plans. Over 30% of all collective pension schemes in the Netherlands are administrated by APG. We manage pension assets of approximately USD 320 billion (as of Dec 31, 2010).

In general pension fund plans in the United States are individual, and based on Defined Contribution (DC). In DB plans, payments to retirees are defined in advance with employers bearing the investment risk. DC plans effectively transfer the risk onto employees by specifying only the contribution to the plan, and making no guarantee about eventual payout in retirement.

The corporate data warehouse of APG Asset Management contains the investments data of Dutch pension funds. The pension assets of these clients are managed both internally and externally (by a number of custodian banks) and include public equity, fixed income, real estate, commodities, hedge funds and private equities.

The data warehouse plays an important role in daily steering (based on un-audited data), formal reporting (based on audited data), and in feeding decision support systems for complex asset management analytics.

The warehouse's user community consists amongst others of business analysts, (client) portfolio managers, controllers, accountants, and risk managers. These people work with SAS[®] Enterprise Guide for analysis and reporting, the SAS[®] Add-In for Microsoft Office for data analysis within Microsoft Excel, and the SAS[®] Information Delivery Portal for canned reports.

The variety of warehouse usage, the diversity of the client requirements, the need for auditable processes, and the complexity of the data streams have prompted us to build a Warehouse management and control system which we present in this paper. In a first section we give an overview of the daily warehouse process, including a sample of the system's requirements, and a demonstration of the main parts of the user interface.

In the following sections we highlight four specific parts of the system, showing you in detail what they do, why we need them, and how we got them to work. The first of these is about interactively generating reports in batch via a stored process. The second is about logging the stored process runs and retrieving the SAS-log in the browser. The third is about delivering reports in the portal and logging their usage, and the last section is about the reports catalogue. By means of conclusion we identify some future work and hint at other possibilities.

OVERVIEW OF THE DAILY PROCESS

REQUIREMENTS

At an aggregated level the operational data warehousing process consists of two parts (Fig. 1):

1. Running a number of ETL jobs that load the corporate data warehouse. This is a classic automated batch-approach. These jobs can be monitored real-time.
2. A mix of automated and interactive processes: data quality monitoring, data management, data integration for updating data marts, multi-client report generation and report publication.

After data quality checks have been executed during the morning process, data for that day is approved by the data quality control team, and the report generation process starts. At the end of the report generation process, elements between reports are being checked for consistency. The final step is the publication of reports to departmental pages/channels.

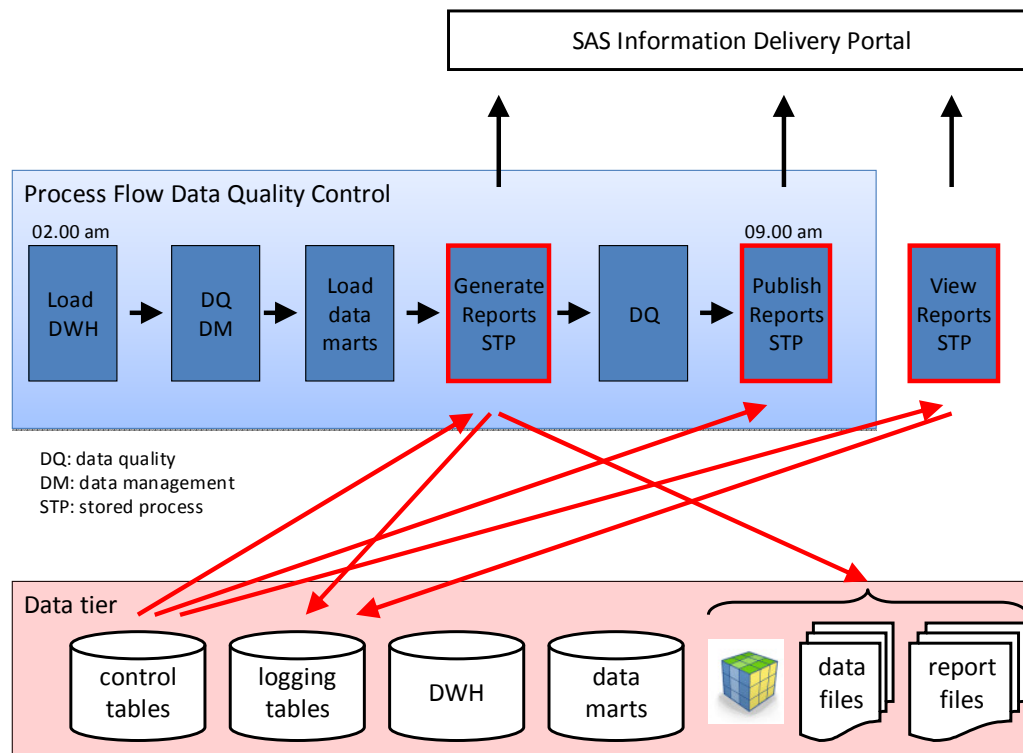


Figure 1. Overview of the data warehouse process; the parts in red are within the scope of this paper.

The ultimate goal of the separation between Data Integration and report generation/publication is to guarantee the accuracy, completeness, consistency and plausibility of the data for DWH users.

Keep in mind that within the context of this paper a report can either mean a report file (pdf, xls or text), or a SAS data set, or a SAS OLAP cube.

To support the data warehouse process a number of SAS stored processes have been developed in our SAS 9.1.3 environment, while taking into account the following requirements as defined by the data quality control team:

1. Support the interactive, independent generation of reports, for a specific date, for one or more clients. The nature of the data creates the requirement to generate report X for client A on the same or on a different day than for client B.
2. Support the interactive publication of reports to one or more departments or to business servers via ftp.
3. The status of the generation and publication-process must be shown in real-time.

- 4. An interface that is easy to understand. Users should be able to use these stored processes with a basic understanding of SAS.
- 5. Users must be able to retrieve reports for a specific date, for one or more clients.
- 6. Reports must be secured at a departmental level.
- 7. Full logging of report and catalog usage, of report generation, and of report publication processes. The logging-data is used for account management purposes, and for periodic monitoring of the report generation and publication processes.

We now show you the most important user interfaces of the stored processes that fulfill these requirements, starting with the interfaces for three stored processes that have been developed for the data quality control team:

GENERATE REPORTS

This stored process is used for the interactive generation of multi-client reports. The end of the page contains the 'run'-button that triggers the generation process.

The Portal page shows the user a list of stored processes that generate the daily report files for every client (only 2 clients listed here for convenience) together with its input parameters. Note that the 'client name' parameter is not an editable input field here, but it is an input parameter for every stored process.

The filter boxes at the top of the screen allow the user to subset the list of stored processes. The checkboxes on the left side of the client name allow the user to check/uncheck all the stored processes for that particular client. The double arrows next to the input parameter values allow the user to 'sync' the entered value in all the other stored processes that have the same input parameter.

Figure 2. Generate Reports Page

PUBLISH REPORTS

This stored process is also used by the data quality control team for the publication of reports to one or more departmental pages (these correspond to secured channels in WebDAV) or to business servers via ftp. Report files in blue indicate reports that already exist in WebDAV. For these report files a new version is being created.

Figure 3. Publish Reports Page

DATA INTEGRATION JOB MONITOR

This application is used for real-time monitoring of data integration jobs. We refer to another paper (Heinsius, 2011) for more info on this topic.

The functionality we've shown so far primarily concerns tools used to support the daily data warehouse process. The following tools have been developed specifically for our DWH user-community (internal employees):

PORTAL WELCOME

This stored process shows the users' 10 last viewed reports and the users' Top10 of most viewed reports during the last 90 days. The page is only available for authenticated users.

Figure 4. Portal Welcome Page

VIEW REPORTS

Used to retrieve reports. The list boxes are used to display all report files for a single client or all multi-client report files for a single report, for a specific date, within a departmental page.

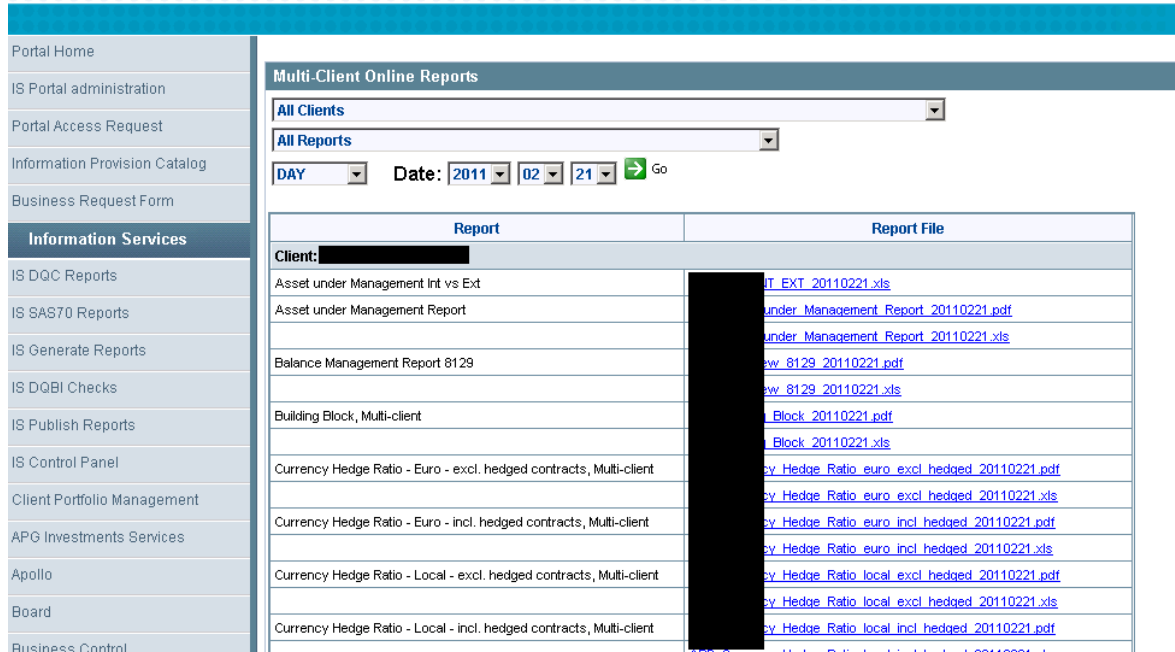


Figure 5. View Reports Page

INFORMATION PROVISIONING CATALOG

The Information Provision Catalog (IPC) is a report catalog that provides information about all the reports that are available in the data warehouse. It is available for all users, including non-authorized users within APG Asset Management.

The catalog allows users to get acquainted with the different types of reports by giving them the possibility to explore the report metadata, view report samples, and subscribe to reports they find interesting. Users can browse through the complete report portfolio or they can search for report information by means of the built-in search functionality.

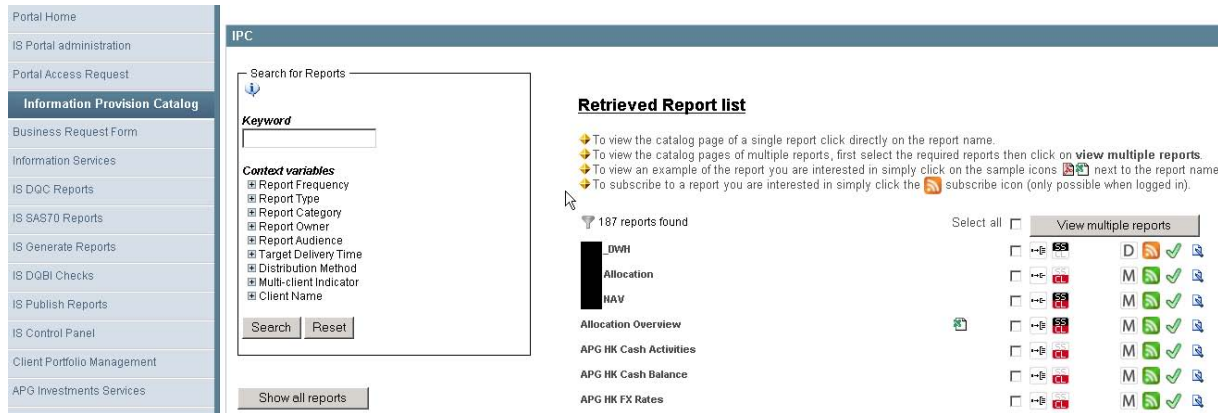


Figure 6. Report Catalog Page

All these tools have been implemented as SAS® stored processes that run on the Stored Process Server (with streaming output) and are leveraged by the SAS® Information Delivery Portal (IDP), using SAS 9.1.3.

The reports themselves are developed in SAS® Enterprise Guide as stored processes running on the Workspace Server, and typically contain parameters to specify ledger/effective date and client name. A single 'reporting' stored process can generate one or more reports. Every single report can have multiple output flavours (xls, pdf, csv, ...).

A graphical overview of this architecture:

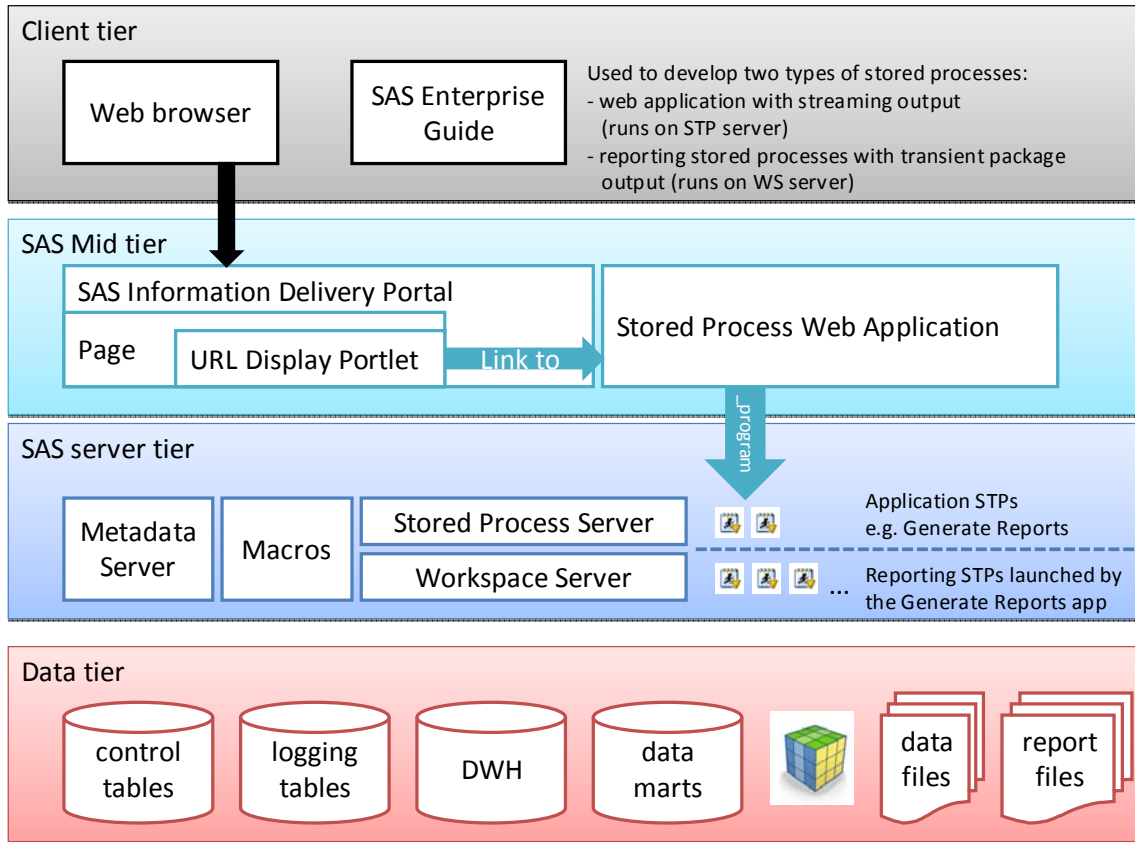


Figure 7. Architectural overview of the system.

We now proceed to describe in more technical detail the following functionality:

- Generating Reports (and logging)
- Publishing Reports
- Viewing Reports (and logging)
- The Information Provisioning Catalog

INTERACTIVELY GENERATING REPORTS IN BATCH VIA A STORED PROCESS

To enable the data quality control-team to run stored processes interactively, in-parallel, and with ease, we developed the "Generate Reports" application that was introduced above (Fig. 2).

At the moment the 'run' button is pressed the actual execution status of the stored processes is shown (Fig. 8). This includes the option to view log files.





IS Generate Multi-Client Reports Shared: In		
Client	Stored Process	Execution Status
Client name 1	Building Block (Client Level)	 Failed with 13 error(s). Elapsed Time: 0:00:16 view log
	Currency Hedge Ratio (Client Level)	 Completed successfully. Elapsed Time: 0:00:15 view log
	Table 1 (Client Level, Allocation Information)	 Running...
	Table 6a (Client Level, Currency Information)	 Running...
	Table 9 (Client Level, Overlay Information)	Queued...
Client name 2	Building Block (Client Level)	Queued...
	Currency Hedge Ratio (Client Level)	Queued...

Figure 8. Status screen after launching the selected stored processes.

To build the "Generate Reports" application we created two stored processes that produce streaming output. They both run on the Stored Process Server and are responsible for the following tasks:

- stored process 1 - "Generate Reports", consisting of two parts:
 - Presents the list of stored processes with their editable input parameters in a structured and ordered way (as in Fig. 2).
This list is created by running the stored process with the 'action' parameter set to the value 'list', and it performs the following steps:

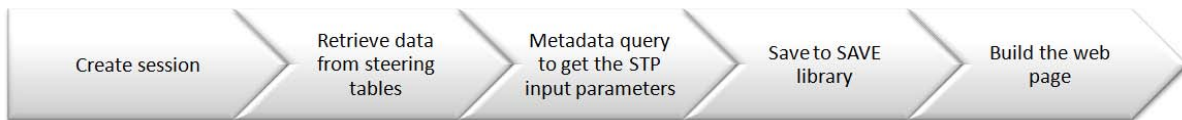


Figure 9. Steps in the 'LIST' part of the "Generate reports" stored process

- Launch the selected reporting stored processes, show the progress, and wait for their status to be streamed back (Fig. 8).
This list is created by running the stored process with the 'action' parameter set to the value 'launch', and it consists of the following steps:



Figure 10. Steps in the 'LAUNCH' part of the "Generate reports" stored process

- stored process 2 - "View log", that retrieves the SAS log file and streams it back to the browser with basic color coding.

The "Generate Reports" stored process has a main macro which is the single statement of the stored process that is initially executed:

```

%macro main;
  %if (%upcase(&action) = LIST) %then
    %list;
  %else %if (%upcase(&action) = LAUNCH) %then
  
```

```

%launch;
%mend main;
%main;

```

Now we will elaborate further on the interesting parts of the steps mentioned in Figures 9 and 10. For readability, the name of each of the steps is printed as underlined text.

LIST

Ignoring session creation, the second step is to retrieve data from control tables which we have developed for this portal application. Based on the contents of the control tables all combo boxes and the list of all active client – stored process combinations are generated.

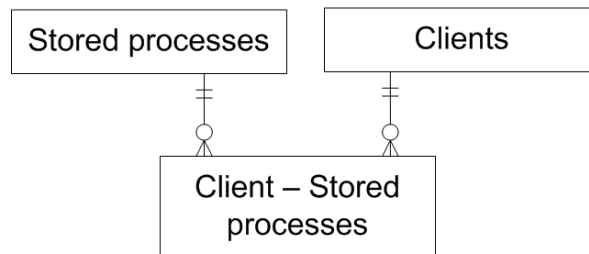


Figure 11. Simplified ER-diagram of the required control tables.

The control tables contain:

- a list of stored processes with some characterizing attributes (e.g. status, frequency, type of output)
- a list of clients with some characterizing attributes (e.g. status, abbreviation)
- a table that links a stored process to one or more clients with a validity period

In this step we also create all IDs/names and the HTML code for almost all the checkboxes in our screen. The trick here is to use a logical naming convention for the checkboxes. This will enable some small but clever JavaScript functions to check/uncheck child checkboxes when for example a parent checkbox is checked/unchecked (we will not go into further detail about these JavaScript functions, but we are happy to share them with you on request).

In the following DATA step the checkbox-variables are created:

```

data work.v_client_stps / view= work.v_client_stps;
  set work.client_stps;
  length sStpId sClientNr $ 8;
  by ClientName StpName;
  if (first.ClientName) then client_nr + 1;
  StpId = _N_;
  sStpId = strip(put(StpId,8.));
  sClient_nr = strip(put(client_nr,8.));
  chkClientName = '<input type="checkbox" '
    !!'name="checkClient_!!sClient_nr!!" '
    !!'id="checkClient_!!sClient_nr!!" '
    !!'onClick="checkUncheckClient(this)" '
    !!'value="true" checked>'
    !!'<label for="checkClient_!!sClient_nr!!"> Client:
    !!ClientName!!</label>';
  tdStpName = '<label for="stprun_'
    !!sStpId!!'_'
    !!sClient_nr!!'>'
    !!StpName!!</label>';
  chkRun = '<input type="checkbox" '
    !!'name="stprun_!!sStpId!!'_'!!sClient_nr!!" '
    !!'id="stprun_!!sStpId!!'_'!!sClient_nr!!" '
    !!'onClick="checkUncheckRunStp(this)" '
    !!'value="true" checked>';
  chkPrint = '<input type="checkbox" name="stpprint_'
    !!sStpId!!'_'

```



```

!!sClient_nr!!" value="true">';
run;

```

As soon as all active stored processes are retrieved it's time for the third step: query the metadata to get the STP input parameters (client name, effective date, ledger date, etc.) We use data step functions to query the metadata. However, to retrieve the input parameters of a stored process you need to know which metadata objects are involved. The stored process parameters are well hidden in the SAS 9.1.3 metadata.

To find these parameters the following hierarchic metadata-structure has to be traversed:

Stored process = ClassifierMap object

Association: Properties (Select the only available "Entity" Property)

Association: PrimaryPropertyGroup (Select the only available "General" PropertyGroup)

Association: PropertySets (Select the only available "EntityAttributeDescriptors" PropertySet)

Association: SetPropertyies → here you can see all parameters of the stored process in a Property object

The following DATA step contains metadata queries to search for the parameters:

```

data stps_with_params;
set stps;
... ❶
spMetadataLocation=''; ❷
numSPs=metadata_getnobj("omsobj:ClassifierMap?@Name ='"!!
strip(StpName)!!'"",i,uriSP);
rc=metadata_getnasn(uriSP, 'Trees', 1, uriTree);
do while(rc gt 0);
rc=metadata_getattr(uriTree, 'Name', nameTree);
spMetadataLocation='/'!!strip(nameTree)!!strip(spMetadataLocation);
rc=metadata_getnasn(uriTree, 'ParentTree', 1, uriTree);
end;
if (upcase(strip(spMetadataLocation)) eq upcase(strip(STP_FOLDER))) then do; ❸
numProperties=metadata_getnasn(uriSP, 'Properties', 1, uriProperty);
if (numProperties gt 0) then do; ❹
numPrimaryPropertyGroups=metadata_getnasn(uriProperty, 'PrimaryPropertyGroup',
1, uriPrimaryPropertyGroup);
if (numPrimaryPropertyGroups gt 0) then do; ❺
rc=metadata_getnasn(uriPrimaryPropertyGroup, 'PropertySets',
1, uriPropertySet);
numSetProperties=metadata_getnasn(uriPropertySet, 'SetProperties',
1, uriSetProperties);
do j=1 to numSetProperties;
numSetProperties=metadata_getnasn(uriPropertySet, 'SetProperties',
j, uriSetProperties);
if (numSetProperties gt 0) then do; ❻
rc=metadata_getattr(uriSetProperties, 'Name', nameParameter); ❼
rc=metadata_getattr(uriSetProperties, 'PropertyName', macvarParameter); ❽
output;
end;
end;
end;
end;
run;

```

Some notes about the code:

- ❶ Initialization of attributes left out for readability.
- ❷ Compose stored process metadata location in the BIP Tree by traversing the metadata trees.
- ❸ Determine if the stored process metadata location corresponds with STP_FOLDER in our control table. If so, then get its parameters.

- ④ Process Properties.
- ⑤ Get process PrimaryPropertyGroups.
- ⑥ Process SetProperties.
- ⑦ Get parameter name.
- ⑧ Get parameter macro variable name.

Note: In SAS 9.2 the stored processes input parameters have been converted to the GroupInfo attribute of the Prompt object which is linked directly to the ClassifierMap object. This GroupInfo object holds all parameters in an XML string with its own 'undocumented' structure. But it's readable and understandable. So you can create an XML map to process the XML string and retrieve the input parameters.

After saving the resulting data set to our SAVE library (not shown) it is time to [build the web page](#) (step 5) with a mix of ODS HTML statements, DATA steps and one single PROC REPORT statement.

```

ODS html file=_webout(no_bottom_matter); ❶
ODS html close;

data _null_;
  file _webout;
  /* many put statements with JavaScript code */❷
  /* many put statements to build the filter boxes form */
run;

data _null_;❸
  file _webout;
  put '<form id="sprunform" action="/SASStoredProcess/do" method="post">';
  put "<input type=""hidden"" name=""_program"" value=""&_program"">";
  put '<input type="hidden" name="action" value="run">';
  put "<input type=""hidden"" name=""_sessionid"" value=""&_sessionid"">";
  stop;
run;

ods html file=_webout(no_top_matter no_bottom_matter);
proc report data=work.steps_with_params style(report)=[cellpadding=3]; ❹
  columns chkClientName tdStpName chkRun chkPrint
           nameParameter inputFieldParameter;
  define chkClientName /group noprint;
  define tdStpName /group width=100 'Stored Process';
  define chkRun /group width=100
    '<label for="selectRun">Run
     <input type="checkbox" id="selectRun" name="selectRun" value=""
     onClick="checkUncheckRun(this)" checked>';
  define chkPrint /group width=100
    '<label for="selectPrint">Print
     <input type="checkbox" id="selectPrint" name="selectPrint" value=""
     onClick="checkUncheckPrint(this)">';
  define nameParameter /display width=100 'Parameter';
  define inputFieldParameter /display width=100 'Value';

  compute before / style=[just=left]; /* print the Select all checkbox */
    length text $ 1024;
    text = '';
    text = ' <input type="checkbox" id="cb" name="cb"
           onClick="checkUncheckAll(this)" value="true" checked />
           <label for="cb">&nbsp;Select All</label><b></b>';
    line text $1024.;
  endcomp;

  compute before chkClientName / style=[just=left];/* print a client header line */
    line chkClientName $300.;
  endcomp;

```

```

compute after / style=[just=left]; /* print the Run button */
  line ' <input type="submit" value="Run Selected Reports" />';
endcomp;
run;
ODS html close;

data _null_;❶
  file _webout;
  put '</form>';
run;

```

Some notes about the code:

- ❶ Create HTML page until the <BODY> tag
- ❷ JavaScript code and the HTML form for the filter boxes with PUT statements (not shown)
- ❸ Start the form and create the action URL to execute when the 'Run'-button is pressed
- ❹ PROC REPORT prints the stored process list to the browser (with all the checkboxes) and adds the run button at the bottom (the lookup from id's to names for stored processes and clients is not shown)
- ❺ Close the form

LAUNCH / RUN

In the run part of our stored process we need to retrieve the selected reporting stored processes and their parameter values (step 1 in Fig. 10). Because the run checkboxes and input parameters have been implemented as form objects their values are passed as parameters to the form and become available as macro variables. Querying the SASHELP.VMACRO dataset view gives access to their values.

```

/* Get the Reporting Stored Processes to Run */
proc sql;
  create table work.stps_to_run as
  select scan(strip(name),2,'_') as sStpId
  from SASHELP.VMACRO
  where upcase(name) like "STPRUN__%"
  and upcase(value) = 'TRUE'
  order by calculated STP_NR
  ;
quit;

/* Get Stored Process Parameters passed in HTML variables from SASHELP.VMACRO */
proc sql;
  create table work.stp_param_values as
  select scan(strip(name),2,'_') as sStpId
  , substr(strip(tranwrd(name,'STPPARM_', ''))
  , index(strip(tranwrd(name,'STPPARM_', '')), '_')+1) as StpParamName
  , strip(value) as StpParamValue
  from SASHELP.VMACRO
  where upcase(name) like "STPPARM__%"
  order by calculated sStpId
  ;
quit;

```

After we found all the selected stored processes and their parameter values, we can build the web page that lists them and that launches them from within the web browser with the use of Ajax.

Let's explain the function of Ajax in this context first. The SAS Stored Process Web Application is a Java Web application that can execute stored processes and return results to a Web browser. The name of the stored process to run is passed along as a value of the `_PROGRAM` parameter.

Here is an example of a URL that executes a stored process:

```
http://yourserver.com:8080/SASStoredProcess/do?_PROGRAM=//Foundation/BIP Tree/Portal/
StoredProcesses/StoredProcessName
```

- /SASStoredProcess/do is the URL of the SAS Stored Process Web Application.

- `_PROGRAM` is the parameter that contains the path and the name of the stored process to run.

Ajax is the perfect way to launch multiple stored processes from within a web browser using JavaScript, because Ajax has the ability to send HTTP server requests in the form of a URL to a web server and capture the response of these requests. This behaviour could also be achieved by using several iFrames in the web page, each of which containing as source the URL of the stored process to run. This approach has one significant drawback: time-out.

We use Ajax in order to control the number of concurrent stored process runs and thus avoid potential time-outs. But where does this time-out issue come from? There is a limit to the number of concurrent HTTP server requests that are allowed in web browsers, and for Internet Explorer 6/7, which is the version we have, this limit is set to two. This means that as soon as a 3rd HTTP server request is fired, it is queued until one of the other two finishes. But if it takes longer than 5 minutes for the 2 HTTP request to finish, the queued request will be timed-out. This means that if we want to launch 3 stored processes in parallel, and the first 2 stored processes take longer than 5 minutes to finish, the 3rd queued stored process will receive a time-out and hence will never be launched. Using Ajax we control when a stored process can be launched, and in our case we launch two stored processes in parallel. Only when one of these two is finished, the next one is launched.

In the following data step we build the stored process URLs that have to be launched and place them in a JavaScript array:

```
data work.stored_processes;
/* ONE RECORD FOR EACH CLIENT-STP COMBINATION THAT NEEDS TO RUN */
set stps_to_run_with_params;
file _webout;
length url stp $512;
by ClientName StpName;
retain url;
if _N_ = 1 then put '<script type="text/javascript">'; ❶

if (first.StpName) then
do; ❷
    url = "&urlSpServer/SASStoredProcess/do"
        !! '?_program=' !! strip(STP_FOLDER) !! '/' !! strip(StpName)
        !! '&client_name=' !! urlencode(strip(ClientName));
    end;
nameValuePair=strip(StpParamName)!!"="!!strip(StpParamValue); ❸
url=strip(url)!!"&"!!strip(nameValuePair);

if (last.StpName) then
do; ❹
    count+1;
    Result = '<div id="result"!!strip(count)!!
        '" style="overflow: hidden; width: 500px; height: 22px;">
        Queued</div>';
    output;
    my_array_num = count-1;
    put 'URL[' my_array_num +(-1) ']=' url +(-1) '";' ; ❺
    put 'RESULT[' my_array_num +(-1) ']="RESULT' count +(-1) '";' ; ❻
end;

if last then do; put '</script>';end;
run;
```

- ❶ We are only creating JavaScript code in this DATA step
- ❷ Start generating the STP URL to be executed
- ❸ Add parameter–name value pairs to the URL
- ❹ Place in Result the DIV-HTML code to where the status message should be streamed back. The HTML DIV elements are inserted in the table cells of the column 'Execution Status' shown in Fig. 8.
- ❺ Fill the JavaScript URL array with URLs
- ❻ The JavaScript array RESULT is filled with the names of the HTML DIV elements that will contain the result messages from the stored processes. Each stored process URL has a corresponding RESULT object which is the

DIV-HTML. This means that when Ajax will launch the n-th stored process by requesting the n-th URL of the URL array, it will write the response of that stored process to the HTML DIV element found on the n-th position of the RESULT array.

We then continue to build our HTML page and stream the list of all selected stored processes to the browser with a 'queued' status.

```
ods html file=_webout(no_top_matter no_bottom_matter);

proc report data=work.stored_processes;
  columns ClientName StpName Result;
  define ClientName      /group width=100 'Client';
  define StpName         /group width=100 'Stored Process';
  define Result          /group width=100 'Execution Status';
  compute after / style=[just=left];
    line "<input type=""submit"" value=""Goback""
        onClick=""javascript: history.go(-1)"" />";
  endcomp;
run;

ods html close;
```

Now we only need to tell the browser to launch the stored processes:

```
data _null_;
  file _webout;
  put '<script type="text/javascript">';
  do i=1 to symget('numParallel');
    /* numParallel = min(max processes in browser, nr of selected stps)*/
    put 'startReport();';
  end;
  put '</script>';
run;
```

The startReport() function is initially called X times, where X is the maximum number of concurrent HTTP requests allowed in a web browser, unless the number of selected stored processes is lower than this maximum. In that case X is set equal to the number of selected stored processes.

The startReport() function uses Ajax to launch one of the stored process URLs found in the URL array, and when the Ajax object finishes processing the URL, it triggers the startReport() function again which results in the next URL in the array being launched.


```

%stpbegin; /* start capturing the output to be send to the portal page */

title;
footnote;
proc print data=MESSAGE noobs label split='~' style(table)={just=l BORDERWIDTH=0};
  var line / style(data)={VJUST=TOP TOPMARGIN=0 BORDERWIDTH=0};
  label line="~";
run;

%stpend; /* close capturing the output and send to the portal page */

```

Note that the post-processing macro contains the %stpbegin and %stpend calls. The %stpbegin macro initializes the Output Delivery System (ODS) to generate output from the stored process. The %stpend macro terminates ODS processing and completes delivery of the output to the client or other destinations.

In the above code snippet you can see that our second stored process “ViewLog” is mentioned in the hyperlink. It streams a log file back to the browser (passed along via the 'file' input parameter). “ViewLog” is a straightforward stored process that runs on the Stored Process Server:

```

*ProcessBody;

%global FILE;

%stpbegin;

options ps=32000;
title;
data _null_;
  length line $ 200;
  infile "&file";
  input;
  file print;
  if (substr(_infile_,1,5) = 'ERROR') then
    line='<font color="red"><b>!!strip(_infile_!!</b></font>';
  else if (substr(_infile_,1,7) = 'WARNING') then
    line='<font color="green"><b>!!strip(_infile_!!</b></font>';
  else if (substr(_infile_,1,5) = 'NOTE:') then
    line='<font color="blue"><b>!!strip(_infile_!!</b></font>';
  else
    line=strip(_infile_);
  put line;
run;

%stpend;

```

EMBEDDING THIS WEB APPLICATION INTO THE PORTAL

In order to avoid a second login prompt for authentication, we registered the stored process as a Portal Web Application, and then used the URL of this application as the source of the URL Display Portlet. For more details see (see SAS Sample 32158).

DELIVERING REPORTS IN THE PORTAL AND LOGGING THEIR USAGE

The report files that are generated by the “generate reports” application are not directly available for report consumers in the portal. Therefore we created a stored process web application to publish the report files to WebDAV (Fig 3.). This application is very similar to the generate reports application (see Fig. 2). In this section we will elaborate on the publish reports application and on the logging of report downloads by users.

PUBLISH REPORTS

The publishing stored process application uses several call routines of the SAS Publishing framework:

- to create packages: call `package_begin`
- to insert files into packages: call `insert_file`
- to publish packages to WebDAV collection URLs: call `package_publish` and call `package_end`

We won't go into the details of all this but it is important to understand how we publish the report files to WebDAV. The 'call `package_publish`' routine is used with the `TO_WEBDAV` publish type as follows:

```
data publish_results;
set files_to_publish;
/* creating packages */
... ❶

/* publishing */
collection_url=channel_url||"/"||package_name; ❷
propertiesPublish = "collection_url, http_user, http_password, if_exists"; ❸

call package_publish(packageId
, 'TO_WEBDAV'
, rc
, propertiesPublish
, collection_url
, "saswbadm"
, "ourpassword"
, 'UPDATEANY'); ❹

...
run;
```

- ❶ In this step we create the packages with a name "<report_name>_<yyyymmdd>" and insert files into them (not relevant here)
- ❷ Here we create the collection URL where we want the package to be published to (e.g.: `http://mywebdavserver:7001/webdavchannel1/report1_20110406`)
- ❸ Here we specify which properties we are going to pass to the `package_publish` call routine
- ❹ Here we publish the package to the WebDAV collection URL and update the package if the package already exists

A file that has been published by inserting it into a package can be downloaded via a URL like e.g.

`http://mywebdavserver:7001/webdavchannel1/report1_20110406/myreport1_client1_20110406.pdf`

LOGGING OF REPORT USAGE

To enable portal users to view reports the “view reports” stored process web application is created (see Fig. 5). This application groups all published packages and their files and presents them with hyperlinks. To retrieve the packages and their files we used the following call routines:

- call `retrieve_package`: returns a package list (some kind of ID for the collection URL of a specific package)
- call `package_first`: uses the package list to return the package ID and the number of files in the package
- call `entry_next`: uses the package ID to list every file in the package (to be used in a loop)

The hyperlink to any report file is constructed as follows:

```
STP_URL = "http://&_SRVNAME:7001/SASStoredProcess/do?!!"
```



```

    "_program=//Foundation/BIP%20Tree/Portal/StoredProcesses/Get_Report"!! ❶
    '&file='!!strip(URL_TO_REPORT_FILE_IN_WEBDAV)!! ❷
    '&log_report_id='!!urlencode(strip(REPORT_ID))!! ❸
    '&log_client_id='!!urlencode(strip(CLIENT_ID))!! ❹
    '&log_channel_id='!!urlencode(strip(CHANNEL_ID))!! ❺
    '&log_requested_filedate='!!urlencode(strip(REPORT_REQUESTED_FILEDATE)) ❻
    URL= '<a href="!!strip(STP_URL)!!'" target="_blank">Report name</a>;' ❼

```

- ❶ We don't link to the report file directly in WebDAV but use an intermediate stored process instead to first log the report file download and only then to redirect the browser to the report file in WebDAV
- ❷ Here we pass the report file URL in WebDAV to the stored process
- ❸ Here we pass other interesting logging information for this report
- ❹ Here we create the final hyperlink for the report file

The intermediate stored process "Get_Report" is a very small but useful logging stored process which enables us to know which reports and topics are frequently requested and which are not. It also identifies reports that are rarely read and perhaps need to be phased out. The following is a very downscaled version of this stored process (we do log more information than what is mentioned below, but we leave it to the reader's imagination to find their own interesting logging bits):

```

%global file; ❶
%global log_report_id;
%global log_client_id;
%global log_channel_id;
%global log_requested_filedate;

*ProcessBody;

/* Create information to be logged */
data report_view_to_log;
length
  view_dt 8.
  user_login $10.
  requested_url $250.
  report_file_name $200.
  report_ext $10.
  report_requested_filedate $20.
  report_id 8.
  client_id 8.
  channel_id 8.
;
format view_dt datetime20.;

  view_dt                = datetime(); ❷
  user_login             = "&_metauser";
  requested_url          = "&file";
  report_requested_filedate = "&log_requested_filedate";
  report_id              = &log_report_id;
  client_id              = &log_client_id;
  channel_id             = &log_channel_id;
  if index(report_file_name, '.')
    then report_ext      = lowercase(scan(report_file_name,-1, '.'));
    else report_ext      = '';
run;

proc append base=LOG.REPORT_VIEW_LOG data=REPORT_VIEW_TO_LOG force; ❸
run;

data _null_;
  file _webout;
  put '<HTML><HEAD>';
  put "<META HTTP-EQUIV='Refresh' NAME='NULL' CONTENT='0;URL=&file'>"; ❹

```

```
put '</HEAD>';  
put '<BODY>';  
put '</BODY>';  
put '</HTML>';  
run;
```

- ❶ Initialize parameters as global macro variables: best practice when writing STPs to prevent WARNINGS
- ❷ Here we create all logging attributes that we want to store in a logging table
- ❸ Here we log the report download in a permanent logging table
- ❹ Here we redirect the browser to the exact WebDAV location, and so release the content to the user

THE INFORMATION PROVISION CATALOG

In this section we highlight the search functionality (Fig. 6), and show how sample reports can be added and viewed through the IPC.

SEARCH

The search functionality lets users search for reports that match a keyword and/or match specific values of context variables (see the left part of Figure 12).

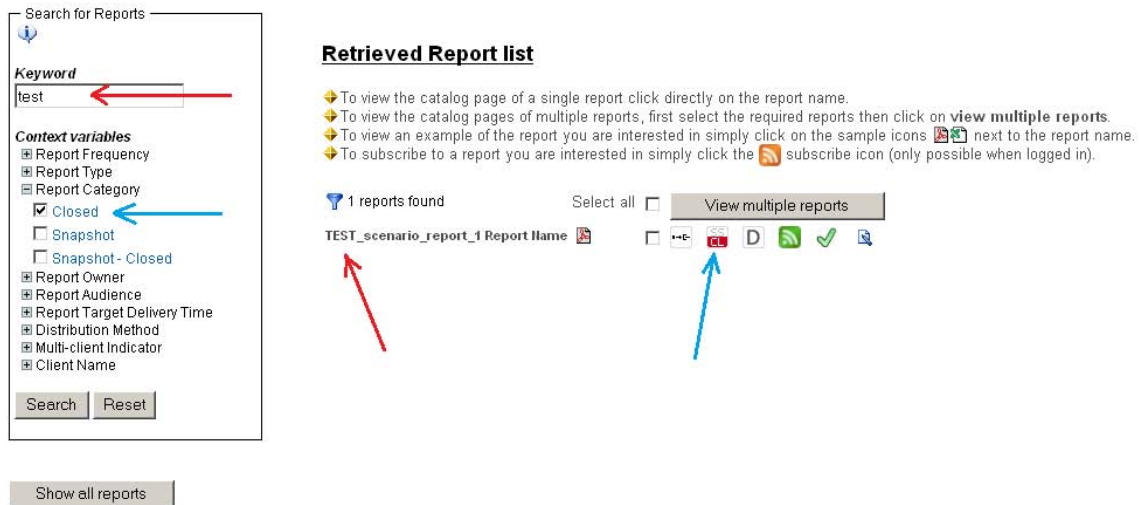


Figure 12. Example of search functionality.

Figure 12 shows an example where the keyword "test" and the context variable 'Report Category' with the value "Closed" result in one report being found that meets the search criteria.

The where clause used to filter the reports consists of a where condition for the keyword and an additional where condition for each context variable.

The where condition for the keyword is built with the following code:

```
%if %SYMEXIST(keywordsearch) %then ❶
%do;
  %if %superq(keywordsearch) ne %then ❷
  %do;
    /* keywordsearch is NOT EMPTY */
    data _null_;
      length keyword $200 myfilter $1000;
      /* to escape the single quote in SQL */
      keyword = tranwrd(uppercase("%SUPERQ(keywordsearch)"), "'", "'"); ❸
      myfilter = "("
        || "index(uppercase(REPORT_CODE),'" || strip(keyword) || "'"
        || " OR index(uppercase(REPORT_NAME),'" || strip(keyword) || "'"
        || " OR index(uppercase(REPORT_PHYSICAL_NAME),'" || strip(keyword) || "'"
        || " OR index(uppercase(REPORT_DESCRIPTION),'" || strip(keyword) || "'"
        || ")"; ❹
      call symputx('whereCondition1',myfilter); ❺
    run;
  %end;
%end;
```

Some notes about the code:

- ❶ 'keywordsearch' is an automatically created macro variable that contains the keyword typed into the search form. If no keyword is typed into the search form then the macro variable will not be created. Therefore we use the

macro function %SYMEXIST to check for the macro variable's existence.

- ② This statement checks whether the 'keywordsearch' macro variable is not blank. Since the keyword text field in the search form allows any character to be typed into it, including text that could have a meaning in the macro code, the macro function %SUPERQ is used to prevent the macro processor from making any attempt to resolve anything that might occur in the resolved value.
- ③ This statement translates each single quote in the keyword into two consecutive single quotes. This is done in order to escape the single quote which is used in the where condition.
- ④ The filter for the keyword is built.
- ⑤ The constructed filter for the keyword is assigned to the macro variable whereCondition1.

The list of available reports displays all distinct reports that match the search criteria from the search utility.



Figure 13. List of available reports.

Figure 13 displays (from left to right) the report name, links to sample reports, and a series of icons indicating client scope (multi-client, client specific, or not client specific), report data category (snapshot, closed, or both), report frequency (daily, weekly, monthly, or quarterly), subscribe status (including ability to subscribe), report status (active or inactive) and a link to the "edit report metadata" screen.

Clicking the report name will open the report definition page which contains detailed information like description, owner, audience, frequency, type, category, delivery time, distribution method, etc.

Some of the icons displayed in the report list, like the report status icon or the "edit report metadata" link, are only available for administrators.

The following piece of code is used to determine if the user is an administrator or not:

```
%let adminGroups=Information Services Authors:Information Services Administrators;①

options validvarname=any;
data _null_;
  length uri uri2 uri3 uri4 name $500 adminGroups $1000 adminGroup $200;
  admin=0; ②
  adminGroups = strip("&adminGroups");
  numGroups = length(adminGroups) - length(compress(adminGroups,':')) + 1;
  do groupNum = 1 to numGroups;
    adminGroup = scan(adminGroups,groupNum,':');
    getGroupId = metadata_getnobj("omsobj:IdentityGroup?@Name = '"
      || strip(adminGroup) || "'",1,uri); ③
    members = metadata_getnasn(uri, "MemberIdentities", 1, uri2); ④
    do loop=1 to members;
      individualMember = metadata_getnasn(uri, "MemberIdentities", loop, uri3);
      rc = metadata_getattr(uri3,"Name",name);
      if upcase(symget('_metaperson')) = strip(upcase(name)) then
        do;
          admin = 1; ⑤
          put name= '*** admin ***';
          leave;
        end;
      end;
    end;
    if admin=1 then leave;
  end;
end;
```

```

        call symput('admin',strip(put(admin,best.))); ❸
run;
%put ADMIN: &admin;

```

Some notes about the code:

- ❶ The macro variable 'adminGroups' contains the SAS metadata group names which represent administrators.
- ❷ The variable 'admin' will have the value 1 if the user is an administrator and the value 0 if not. Here we initialise the variable with value 0.
- ❸ This statement retrieves the metadata identifier of the specified group.
- ❹ This statement retrieves the URI that points to the members list of the group and the number of members found in the list.
- ❺ If the name of the member corresponds with the name of the user ('_metaperson') then the user is an administrator and the 'admin' variable is set to 1.
- ❻ Finally the admin value is assigned to the macro variable 'admin'.

In the code we then use the macro variable 'admin' to decide whether or not to display some screen elements .

VIEW SAMPLE REPORTS

Each report can have one or more sample reports associated. Sample reports are stored in a WebDAV folder and the list of available sample reports is generated with the following data step code:

```

/* Get list of sample reports available in WebDAV folder */
filename WBDVdir sasxbamw
"http://&_SRVNAME:8300/sasdav/Portal/IPC/Sample_reports/" DIR user="&webDAVUser"
pass="&webDAVpw"; ❶
data sample_reports;
  length reportPhysicalName reportFile $1000;
  keep reportPhysicalName reportFile;
  dirId = dopen("WBDVdir"); ❷
  repCount = dnum(dirId); ❸
  do i = 1 to repCount;
    reportFile = urldecode(dread(dirId,i)); ❹
    reportPhysicalName = scan(reportFile,1,'. ');
    output;
  end;
  rc = dclose(dirId); ❺
run;
filename WBDVdir clear;

```

Some notes about the code:

- ❶ In the FILENAME statement, SASXBAMW specifies the access method that enables accessing remote files by using the WebDAV protocol. The DIR option enables accessing directory members.
- ❷ DOPEN opens the WebDAV directory.
- ❸ DNUM returns the number of members in the WebDAV directory.
- ❹ DREAD returns the name of i-th WebDAV directory member. The URLDECODE function decodes the member name using the URL escape syntax. (The URL escape syntax is used to hide characters that might otherwise be significant when used in a URL)
- ❺ DCLOSE closes the WebDAV directory.

The link to each sample report is then created with the following code statement:

```

data sample_reports;
  set sample_reports; ❶
  ...
  sampleReportUrl = '<a href="' ❷
  || "&_URL?_program=/BIP_Tree/Portal/StoredProcesses/Standard/
Public/MC_Get_Report" ❸

```

```

|| '&file=' ❹
|| "http://&_SRVNAME:8300/sasdav/Portal/IPC/Sample_reports/" ❺
|| strip(reportFile) ❻
|| '" target="_blank">'; ❼
...
run;

```

Some notes about the code:

- ❶ Read the list of available sample reports that was generated with the previous piece of code.
- ❷ Create a variable 'sampleReportUrl' that contains the URL to the sample report.
- ❸ The reserved macro variable _URL refers back to the SAS Stored Process Web Application. The stored process 'MC_Get_Report' is used to log and open the sample report: this logging mechanism allows us to report on the most used sample reports.
- ❹ 'file' is the input parameter of the stored process 'MC_Get_Report' which contains the URL of the sample report to be logged and opened.
- ❺ The WebDAV folder where the sample reports are stored. It makes use of the automatic macro variable _SRVNAME.
- ❻ The name of the sample report found in WebDAV with the previous piece of code.
- ❼ This option indicates that the sample report should be opened in a new web browser window.

ADD SAMPLE REPORTS

Sample reports can be added via the edit report metadata screen shown in figure 14.

Search for Reports

Keyword

Context variables

- Report Frequency
- Report Type
- Report Category
- Report Owner
- Report Audience
- Report Target Delivery Time
- Distribution Method
- Multi-client Indicator
- Client Name

Search Reset

Show all reports

Attributes that cannot be modified:

Report Name	Building Block, Multi-client
Report Code	BUILDBLOCK
Report ID	95429096
Report Physical Name	Building_Block
Report Directory	.
Report WebDAV Folder	Building_Block
Report Schedule	DAY
Last updated by	kd13075
Last updated on	13DEC2010:15:31:05

Attributes that can be modified:

Sample reports

Building_Block.pdf

Add sample report

Report Description

The report is used to monitor the composition and development of Client investment

Figure 14. The edit report metadata screen.

Clicking on the 'Add sample report' button will open a popup window from where a sample report can be selected to be added to the IPC.

Starting with SAS 9.2 it is possible to upload files from the web browser to the SAS server via the stored process web application, but since we used SAS 9.1.3 at the moment of developing this application, we had to use an alternative solution.

Figure 15 shows the process we defined to add sample reports to the IPC:

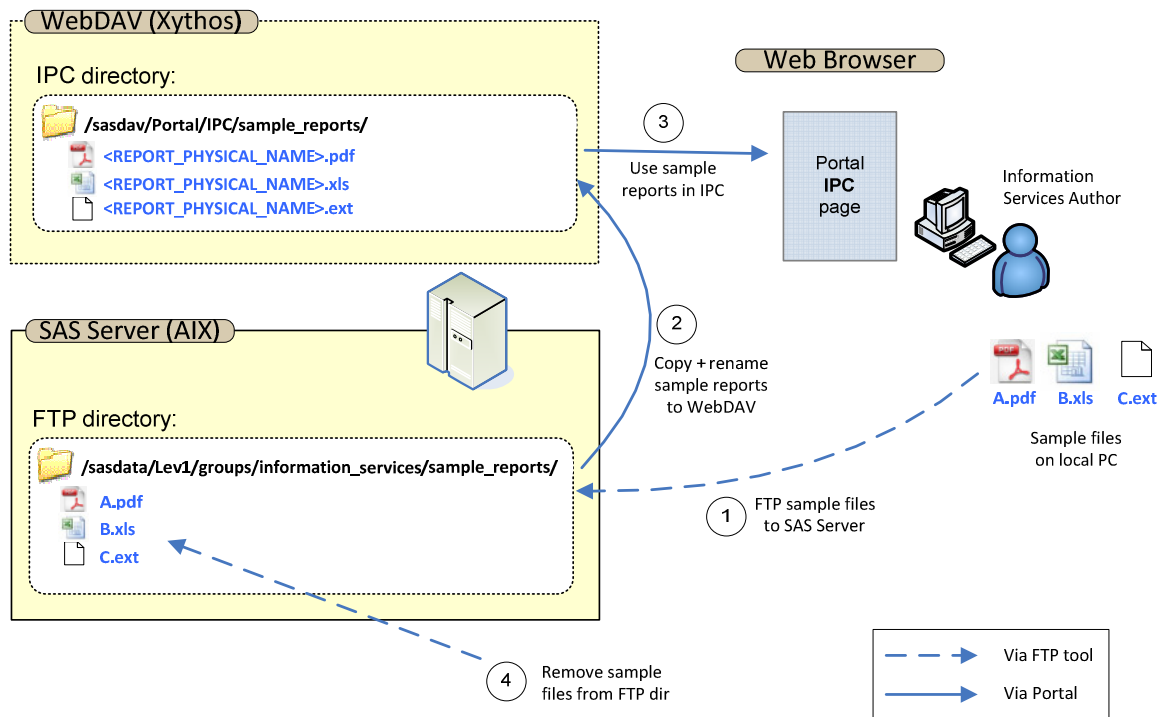


Figure 15. The add sample reports process.

In this process an Information Services Author (a user with admin rights) copies the report file to the sample reports directory on the SAS server (step 1). This is done manually with an FTP tool. All kinds of file types (PDF, Excel, Word, ...) can be used as sample reports.

Afterwards the Information Services Author will add the sample report to the IPC using the 'Add sample report' button in the edit report metadata screen (Fig. 14). This will result in the sample report being copied to the WebDAV storage environment (step 2). After that, the reports are available in the IPC (step 3). The temporary report files can now be removed from the FTP directory on the SAS server (step 4).

The code that copies the sample report files to WebDAV (step 2 in Fig. 15) is the following:

```
filename fsource
"/sasdata/Levl/groups/information_services/sample_reports/&reportFile2"; ❶
filename ftarget sasxbamw
"http://&_srvname:8300/sasdav/Portal/IPC/Sample_reports/&newReportFile..%scan(&reportFile2,2,.)" debug recfm=S user="&webDAVUser" pass="&webDAVpw"; ❷
/* Write binary file to WebDAV */
data _null_; ❸
  infile fsource recfm=N;
  file ftarget ;
  input byte $char1. @;
  put byte $char1. @;
run;
```

Some notes about the code:

- ❶ The file reference 'fsource' points to the source sample report file on the SAS server.
- ❷ The file reference 'ftarget' points to the target sample report file on the WebDAV server.
- ❸ This data step reads the source file from the SAS server and writes it to the WebDAV server.

CONCLUSION

The tools we presented in this paper have been developed as stored process web applications. With the power and flexibility of these applications we have been able to create interactive web applications that support a high controlled data warehouse process and that also supported the transition to a multi-client company.

In the setup of the current stored process web applications the coding for data processing (SAS) and presentation to the web browser (HTML/JavaScript) is mixed and rather complicated. This implicates that the maintenance of these applications can be improved. To accomplish this we are loosening the link between the interactive features needed in the browser and the required data processing. By using documented JavaScript libraries (e.g. Ext Js from Sencha®) we are able to develop rich internet applications, that have a more compact and structured code, and that are easier to maintain.

REFERENCES

Heinsius B., "Data Integration Monitor", *Proceedings of the SAS Global Forum 2011 Conference*, Las Vegas, paper 138, 2011

SAS Sample 32158: How to Create a URL Display Portlet that Displays the Output of a SAS® Stored Process

RECOMMENDED READING

Philip Mason, Building Interactive Web Applications using Stored Processes
SAS Global Forum paper 031-2009

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Ralf Vaessen
APG Asset Management
PO Box 2889
6401 DJ Heerlen
The Netherlands
Email: ralf.vaessen@apg-am.nl

Danny Pannemans
BI'N'GO bvba
Vogelsancklaan 329
3520 Zonhoven
Belgium
Email: danny@bi-n-go.be

Juan Quesada
QSD bvba
Jan Frans Willemslaan 30
3550 Zolder
Belgium
Email: juan.quesada@qsd.be

Koen Vyverman
SAS Institute B.V.
Flevolaan 69
1272PC Huizen
The Netherlands
Email: support@snl.sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.